Minutes of the 36th ARG Meeting

31 October, 1-2 November 2008

Portland, Oregon, USA

Attendees: Steve Baird, John Barnes, Randy Brukardt, Brad Moore, Erhard Ploedereder, Jean-Pierre Rosen, Ed Schonberg, Tucker Taft, Bill Thomas (except Sunday).

Observers: Greg Gicca (except Sunday), Matt Heaney (except Friday and first hour on Sunday).

Meeting Summary

The meeting convened on 31 October 2008 at 09:15 hours and adjourned at 11:45 hours on 2 November 2008. The meeting was held in a conference room at the University Place Hotel in Portland, Oregon, USA. Despite taking advantage of the extra hour provided by the end of Daylight Saving Time on Sunday morning, the meeting covered only about half of the entire agenda, although it did cover all of the ASIS agenda.

AI Summary

The following AIs were approved with editorial changes:

AI05-0050-1/05 Return permissions are not enough for build-in-place (7-0-0)
AI05-0054-2/04 Variable views of constant objects (7-0-1)
AI05-0067-1/06 Objects that are built in place (6-0-1)
AI05-0095-1/02 Address of intrinsic subprograms (8-0-1)
AI05-0099-1/02 The tag, not the type, of an object determines if it is controlled (8-0-1)
AI05-0100-1/02 Placement of pragmas (8-0-1)
AI05-0101-1/03 Remote functions must support external streaming (5-0-2)
AI05-0106-1/02 Representation items are not allowed on generic formal parameters (6-0-3)
AI05-0108-1/01 The incomplete view from a limited view does not have a discriminant part (8-0-0)
AI05-0109-1/01 Impossible check in S'Class'Input (8-0-1)
AI05-0118-1/01 How do parameter associations associate? (8-0-1)
AI05-0120-1/01 Is the current instance of a protected object a constant? (8-0-1)
AI05-0126-1/01 Dispatching when there is no inherited operation (8-0-0)

The intention of the following AIs was approved but they require a rewrite:

AI05-0116-1/01 The value of Alignment for a class-wide object (7-0-2)
AI05-0128-1/00 "/=" is a primitive operation (9-0-0)

The following AIs were discussed and assigned to an editor:

AI05-0001-1/02 Bounded Containers and other container issues
AI05-0123-1/01 Composability of equality
AI05-0127-1/01 Locale consideration for To_Upper and To_Lower

The following AIs were discussed and voted No Action:

AI05-0054-1/01 Variable views of constant objects (7-0-1)
AI05-0054-3/01 Variable views of constant objects (7-0-1)
AI05-0054-4/01 Variable views of constant objects (7-0-1)
AI05-0121-1/01 Opening a non-existent file in Append_File mode (7-0-1)

The following SIs were approved with editorial changes:

SI99-0007-1/03 Add support for new object oriented prefix notation (9-0-0)
SI99-0019-1/02 Add method to check if a name is an implicit dereference (9-0-0)
SI99-0021-1/04 Obtain representation clauses based on defining identifiers, not declarations (8-0-1)
SI99-0038-1/01 Generic units need to be included in Is_Body_Required (9-0-0)
SI99-0039-1/01 Change representation clauses to aspect clauses (9-0-0)
SI99-0041-1/01 Adopt UTF-16 for ASIS (6-0-3)

The intention of the following SIs was approved but they require a rewrite:

SI99-0024-1/10 Provide a semantic model in addition to existing syntactic model (9-0-0)
SI99-0037-1/01 Review std for problematic wording (9-0-0)
SI99-0040-1/00 Various questions (9-0-0)

Detailed Minutes

Meeting Minutes

The minutes were approved unanimously without additional changes. (John and Brad had sent some to Randy before the meeting.)

Date and Venue of the Next Meeting

The next meeting will be in Tallahassee FL, February 20-22, 2009. This is immediately after the PRG meeting. Brad will talk to Ted Baker to make sure he knows that we are coming (and to make other arrangements as needed). After that, we'll meet at Ada Europe, Brest, France, June 12-14, 2009 (WG9 is Friday morning, June 12).

WG 9 Action

WG 9 has tasked us with creating a new Amendment, to be delivered to WG 9 by the end of 2010. The intent is to have a smallish change that would effectively be Ada 2005 Release 2 (as opposed to a major new version). We would expect to cut off consideration of new AIs (at least new Amendment AIs) by Brest (June 2009). Randy will develop the schedule requested by WG 9 and circulate it.

ACATS Test Objectives

Randy had distributed a new Test Objectives document before the meeting. We review the document to ensure that all of the objectives are technically justifiable.

Objective 3-07: Jean-Pierre notes that the test should test recursive declarations (function returning anonymous access-to-function returning anonymous access-to-function...) [This objective was already marked as being incompletely tested; this suggestion was added to the remaining items to test – RLB.]

Objective 3-08: Tucker asks if this should include conventions. Randy replies that conventions would be tested under 6.3.1(13.1/2), which has not yet had objectives developed.

Objective 3-12: "Anonymous" is spelled wrong.

Objective 3-22: "...of a[n] component..."

Objective 3-24: "...of a[n] use type"

Objective 6-08: "..an {an} .."

Accept test objectives: 7-0-2

Copyright issues for ASIS:

Jean-Pierre Rosen will check with AFNOR to see if they can publish the standard cheaply.

Erhard suggests asking Jim Moore to ask ISO to determine whether the copyright covers the specifications, as that being the case is nonsense. If that was true, vendors could not distribute products including the interface itself without violating the copyright. (That would be true whether or not they purchased copies of the standard from ISO.) In that case, an interface standard would be worse than useless – it would actively discourage use of the standard.

Thanks

Thank our hosts (SIGAda) for the fine accommodations, especially the large amount of break food.

Thanks to Adam Beneschan for giving us such fun AIs to work on. (This was unanimously adopted while working on AI05-0126-1, but it applies to the entire body of work that he has submitted.)

Old Action Items

Greg has several action items on hold until the final ASIS version. Greg tried to do Jean-Pierre's example, but the programs did not seem appropriate. Jean-Pierre will look for other examples. Bob did not do an example (which did seem appropriate for the semantic subsystem), Ed will try to get Sergey to do that. Randy did not do AI00-0049-1, the extension of Ada.Directories.

New Action Items

The combined unfinished old action items and new action items from the meeting are shown below.

Steve Baird:

- AI05-0123-1
- Create an AI to fix the visibility bug in 10.1.1(12.2/2) (see the discussion of SI99-0024-1).

Randy Brukardt:

- AI05-0049-1
- AI05-0116-1
- SI99-0024-1
- Create a proposed schedule for the development of Amendment 2 and circulate it.
- Just before submitting the ASIS standard, check that the boilerplate sections (Foreword, 1.2 Normative References) have the current boilerplate.

Editorial changes only:

- AI05-0050-1
- AI05-0054-2
- AI05-0067-1
- AI05-0095-1
- AI05-0099-1
- AI05-0100-1
- AI05-0101-1
- AI05-0106-1
- AI05-0108-1
- AI05-0109-1
- AI05-0118-1
- AI05-0120-1
- AI05-0126-1
- SI99-0007-1

- SI99-0019-1
- SI99-0021-1
- SI99-0038-1
- SI99-0039-1
- SI99-0041-1

Bob Duff

- Create an ASIS example tool to find return-by-reference incompatibilities (preferably both using and avoiding the new semantic subsystem). Possibly reassigned to Sergey Rybin.

Greg Gicca:

- SI99-0037-1
- Add newly ARG approved SIs to the draft ASIS standard.
- Renumber clauses to eliminate so-called hanging paragraphs (at end of standard creation process, not now; see ASIS standard format in the Paris minutes)

Matthew Heaney:

- AI05-0001-1

Pascal Leroy:

- Create AI to combine wording in AI05-0006-1 with other parts of the language (from his editorial review).

Brad Moore:

- AI05-0127-1

Jean-Pierre Rosen:

- SI99-0040-1 (split into three SIs)
- Convert an existing ASIS example tool to use the semantic subsystem.
- Check with AFNOR to see if they could publish the ASIS standard at a reasonable cost.
- Create an AI to propose package(s) to convert between UTF-8 / UTF-16 and Wide_String / Wide_Wide_String (see discussion of SI99-0041-1).

Tucker Taft:

- AI05-0128-1

Bill Thomas:

- Create an SI to clarify the use of contexts in normative wording (not an informational Annex). The SI needs to correct the wording of several clauses (see the discussion of SI99-0037-1).
- Create an example of the semantic subsystem for adding to Section 2 (see the discussion of SI99-0037-1).

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into ASIS Issues (SIs), Ada 2005 non-amendment AIs, and Ada 2005 amendment AIs. The AIs and SIs are presented in numeric order, which is not necessarily the order in which they were discussed. Votes are recorded as “for”-“against”-“abstentions”. For instance, a vote of 6-1-2 would have had six votes for, one vote against, and two abstentions.

If a paragraph number is identified as coming from the Amendment (with a /2 version), the number refers to the text in the Final Consolidated AARM. Paragraph numbers in earlier drafts may vary.

Detailed Review of ASIS Issues

SI99-0007-1/03 Add support for new object-oriented prefix notation

This is similar to the way that operator infix notation is checked in ASIS.

The term is "prefixed view", perhaps this should be Is_Prefixed_View. "View" means semantics, so leave the function name as it is.

In the Note in 17.30, it should say "prefix[ed] notation". Jean-Pierre notes that this is a question to the group. He wonders if there is a way to normalize these expressions. There is no problem here, because there is an implicit dereference node (see SI99-0019-1), so it is OK, it will be normalized to use that node.

Move the note into the discussion and make the dependence on SI99-0019-1 explicit.

"one on one" should be "one-to-one" (in both revised paragraphs).

Steve asks about renames of a prefixed view. Jean-Pierre says that normalized view of a renamed item matches the specification of the item (which has one fewer parameter for a renames). That probably should be mentioned somewhere (perhaps the discussion section of this SI).

Approve SI with changes: 9-0-0.

SI99-0019-1/02 Add method to check if a name is an implicit dereference

Tucker just added a new kind of expression to represent an implicit dereference, that can be used anywhere that the explicit dereference can. The only place he could find where explicit dereference was explicitly used was in 17.9.

Jean-Pierre responds to a question that this is consistent with the way ASIS works, it would be an improvement.

Jean-Pierre asks that Implicit_Dereference follow Explicit_Dereference so they could be a subtype. Tucker thought that these were alphabetical, but that is not true for this enumeration, so Jean-Pierre is right and the change should be made.

Approve SI with changes: 9-0-0.

SI99-0021-1/04 Obtain representation clauses based on defining identifiers, not declarations

Randy notes that "representation clauses" should be "aspect clauses"; that's a separate SI (SI99-0039-1) to change the rest of the Standard. Jean-Pierre notes that he took advantage of that name change to simplify the names of the routines.

Remove extra !wording.

In question: "Clearly, a representation clause does not apply to a declaration, but to an [element]{entity}. To avoid problems with views etc., it is suggested to have a similar query applying to a defining name."

What should the name of the new pragma routine be? We can't use the same aspect trick with it.

Corresponding_Pragmas_of_Entities

Corresponding_Pragmas_of_Defining_Name

Associated_Pragmas - Jean-Pierre says that ASIS semantic functions (in the syntactic subsystem, which this is) should start with Corresponding.

Applicable_Pragmas - Steve and Erhard don't like this.

Corresponding_Aspect_Pragmas (as this is only asking about aspects) That is well liked, but does it return pragma Priority? (That's not an aspect.) But does it return Suppress and other such things?

What does it mean for a pragma to apply to an entity? Jean-Pierre doesn't really know. That's only well-defined for aspect pragmas. (We made the meaning of Suppress of an entity obsolescent because it was too hard to answer that question.)

We should make it clear that this only applies to aspect pragmas, then we don't have to answer whether Suppress applies. Do inherited ones apply? We want to do the same as aspect clauses (this is really part of that). So use the wording from the Aspect clauses, appropriately modified for pragmas. And use the name Corresponding_Aspect_Pragmas.

Approve SI with changes: 8-0-1.

SI99-0024-1/10 Provide a semantic model in addition to existing syntactic model

Should these be interfaces? It would make it much easier to implement. You would want a root type that implements some (or many) of the common concrete operations, but you can't do that with an abstract type hierarchy (that would be multiple inheritance, which is not allowed). These thus should be interfaces.

The intent is that no null value is needed (the Holder container can be tested for empty if needed). This should be explained somewhere.

There should be some text somewhere (in the AASIS, at least) describing the intended implementation.

Delete the question in xx.x.1.1 (it was answered last time).

xx.x.1.2: Expr_Denoting_View seems wrong, because not every view is an expression (it could be a subtype or a subprogram or a package or ...). Even ASIS, with its loose view of categories, is stricter than that. Either it needs to say when it doesn't work, or it needs a more general name.

```
function Corresponding_Element (V : View)
              return Asis.Element is abstract;
```

or possibly "Element_Denoting_View". This gives you the reference that created this view, it is the usage of the view, not necessarily the declaration of this view. This should be explained in detail in the wording, preferably with examples.

This works because ASIS doesn't have typing on elements, so we only need this one function.

xx.x.1.3 Convention_Identifier should return Wide_String (Identifier only works if the value is actually in the ASIS context).

xx.x.1.9: This is fine, drop the question.

xx.x.1.10: "Function Declaration returns that [view]{declaration}, or..."

xx.x.1.11: "Type View_Holder allows the declaration of an uninitialized variable (including a component) that can hold a View.

xx.x.1.12: Delete the function Equal and the parameter, because AI05-0071-1 eliminates the need for it. The rest should say: "Type View_Vector allows the declaration of a list of Views."

xx.x.2.1: There is a fragment about Program_Unit_Vector, it should be deleted.

xx.x.2.4: Use the vector wording and other changes from xx.x.1.12.

xx.x.3.2: Is_Base_Subype is the wrong name; this has nothing to do with the Ada notion of base subtype. Randy says that this is the italicized T of the standard. Tucker suggests Is_Unadorned_Subtype. Rewrite the wording to be more consistent with 4.5.1(4.d-f).

xx.x.3.4: Ultimate_Ancestor should not go through private types. The minutes and the note are wrong. It's possible to note that you've landed on a private type, call Full_Type, and continue. But if you punch through everything, you can't undo that.

Steve Baird wonders if you can get different answers for a parent. For example, if you can see that A is an array, and B is derived from A, but you are not allowed to see that B is an array. ASIS should do whatever Ada does, no matter how stupid.

Again, the view you have may have less information than the full type.

xx.x.3.5: The note is OK. Invert the sentences for Complete_View:

"If the given subtype is an incomplete view, Complete_View returns the completion of the given subtype. Otherwise, Complete_View returns the given subtype."

But does the incomplete view from a limited view always have a completion? What if the incomplete view is of an incomplete type? What if the full view of the limited view is not in the ASIS context?

We should have an exception to be raised if anything is missing (not available in the ASIS context) when it is queried for. That goes at the very top of the interface, and we'll use general wording to define it and won't repeat that everywhere.

We still need to reword Complete_View to handle incomplete views from limited views.

xx.x.3.6 Add Implementation_Defined_Aspect, and add Aspect_Identifier which returns Wide_String.

We need to move aspects up a level, so that they can be queried for all entities. Then add the other aspects to them.

xx.x.4.3 This note is OK.

xx.x.5.5 has the wrong title.

xx.x.5.6 Leave this as is and delete the question. Merge xx.x.5.5 and 5.6 "Family index queries". (Same with xx.x.5.3 and xx.x.5.4.)

xx.x.5.8 should return a Wide_String.

xx.x.8.3 remove the "sorry Bill".

xx.x.8.4: Returning an empty list is fine here.

xx.x.8.5: All of these comments can go, because Is_Prefixed_View is False for a task or protected object.

xx.x.9.1: It can be a renaming. The "non-renaming" can go away; it should be just as "...a view of a package".

xx.x.9.5: Drop the question.

Steve notes a bug in the Ada Standard, in 10.1.1(12.2/2) We need to add "in the visible part" to this paragraph; we don't want packages nested in the private part to show up in the limited view. Steve Baird will take an action item to fix this.

xx.x.9.7: Drop the question.

xx.x.9.8: The first question is fine, just remove it.

The second question needs some work. Instances are expanded, so you get to the body of the instance through the instance. But we don't seem to have a function for that purpose.

In xx.x.2.1, we should add Expanded_Body to get the body of an instance.

To get the body of a package, call Defined_Region (Package_View) and then you can get the parts.

Stub operations are in xx.x.2.2. Drop this question.

xx.x.10: This is fine, drop this question.

xx.x.10.1: Drop non-renamed again from the wording, and remove the question.

xx.x.11: Is_Imported is generally available, drop that question. The second question is the one Jean-Pierre raised earlier (see xx.x.1.3) and we have fixed that.

xx.x.11.1: Drop non-renames.

xx.x.11.2: This function is duplicated; the user can query Aspect_Clauses to get this. So drop this (but add a note).

xx.x.12: We do not want any of these extra operations at this point. They are more syntactic. Drop the question.

xx.x.13.1, xx.x.14.1, xx.x.15.1: All should return classwide types, add 'Class.

xx.x.15.1: Randy wonders what this returns for an operation that is not a name, such a membership or short circuit expression. Tucker claims this should say: "Returns the view of an entity denoted by Expression." But a short circuit is not an entity. Erhard suggests "Returns the view that describes the semantic meaning of the Expression."

Approve intent of SI: 9-0-0.

Randy will manage this, and farm out work if needed to Tucker, Steve, and Bill.

SI99-0037-1/01 Review std for problematic wording

The subject should say "standard" not "std".

Extra period in the summary. Answer should be "(Yes.)" in question.

"For{e}word section".

Obsolescent features are not for information only, so that question should be deleted.

The Foreword is ISO boilerplate. Randy should check that this boilerplate is correct at the last minute (as with other changes to the Standard).

The question about the Introduction is covered in SI99-0030-1, so it doesn't need to be covered here.

1.2 needs to have the full name of the Corrigendum and Amendment.

No example uses obsolete features. Bill would like an example of the semantic subsystem into Section 2. Good idea: he has just got himself an action item.

Global: We've previously discussed this at the WG 9 level and earlier in this meeting, so delete this question.

Underscore problem. Word seems to be hiding the underscores; changing the display settings might show them, and they definitely print properly (including in the PDFs). So don't mess with that – it's more likely to break something than fix it. Delete this question.

Adding the (E) was done in SI99-0030-1, so it doesn't need to be done here.

Fix "before before", of course. But there needs to be references of where these errors are.

The change to 3.8 is a good idea and should be done.

3.1 should be 3.10. The sentence should just say "and pragmas" (and not talk about the specific pragmas).

3.9.22: No such thing as a "use subprogram", so we don't need any such enumeration.

3.12.4: Definitely fix the bug. Pascal wanted to reword "withs". "A mentions B and B mentions C in with_clauses" is a better replacement. In the latter case, "A mentions B and B mentions C and C.Y in with_clauses.and C does not have a with_clause".

Section 4 is a repeat of an earlier item.

10.13: There is a reference to The_Context, but one of the forms does not have that parameter. After much gnashing of teeth, we decided that the definition is for the second form (with the The_Context parameter). The first form is defined in terms of the second. We should rearrange the text so that there is no confusion (moving the first declaration must later).

Jean-Pierre wonders if we should make it obsolescent. But that would be incompatible.

We discuss the meaning of "context" for a long, long time. They appear to be "environments". Bill will look at the old notes in order to determine what was meant. How could you compare two elements? Normally, they would simply be completely different. Tucker moves to split out this into a separate SI, and then fix the wording about cross-context operations. (Those cases where there is a The_Context parameter and a version without it.) Bill will take this SI to try to figure out an answer. Wording should be fixed 10.11, 10.12, 10.13, and 15.26 at least.

On Saturday, Bill notes that D.4.9 tries to explain the meaning of The_Context. We read this informational annex subclause. It clearly needs to be part of the normative specification. One option is to eliminate the context parameters (by making those versions obsolescent). Otherwise, we need to describe the semantics better (and normatively). Ed says that the concept seems to make no sense for a source-based implementation like GNAT. It would be impossible for the same source to appear in different contexts, as their ASIS implementations processes it on demand, and of course different source is just different. He thinks that it makes more sense for a binary-library based implementation, but Randy is somewhat dubious – different compilations of the same source could depend on different other units and thus end up with a completely different meaning. It makes just as much sense to imagine it to work between different, but similar source files (with the same library unit name, for instance), possibly using a project management facility. It's hard to imagine precisely what could be done with this portably (it's all implementation-defined anyway). Bill will have to decide if this is worth keeping given the effort to normatively define what it does.

10.25, 10.27: Fix the package prefix.

11.3: Replace 86_400.0 with Ada.Calendar.Day_Duration'Last, the value is too magic.

12.1: The example text in the fixed width font should be in the regular font. Fix the typo.

12.2: Just need a sentence to describe this. Greg will check to see if either 12.3 or 12.4 should specify that this is returned.

13.1: This appears to be a junk sentence, so delete it.

13.3: Remove the second sentence, because the implementation permission covers the same ground.

Move the paragraph about Standard above the implementation permissions. "Whether package Standard has physical existence is implementation-defined." Physical is awful. Tucker would like to delete the existing paragraph (which is far more general than just this place) and put an Implementation Permission into 3.12.3, that says that for units with an Origin of other than Application_Defined_Unit, the results of some ASIS queries is implementation-defined. Better have an actual list of what it meant by "some queries". Greg has the short stick on this one.

13.6: Fix typo (wrong parameter name): Compilation_Unit should be Element.

15.3: This is an ancient function, so leave it.

15.8, 15.9: This appears to be overtaken by events (there doesn't appear to be an error here).

15.25: Yes, do that.

Looking at all of the routines with problematic descriptions of parameters or results:

15:27: Put A_Pragma above A_Declaration.

15.28: Remove extra nesting if there is only one choice as here. Same in 15.29 (parameter) and 15.32 (parameter).

15.36: drop "one of" from Clause_Kinds (only one element).

15.47 is OK as is.

16.4: drop "one of" from Formal_Type_Kinds.

16.13: Change to:

Definition expects an element that has one of the following Definition_Kinds:

A_Constraint that has the following Constraint_Kinds:
A_Delta_Constraint

A_Type_Definition that has one of the following Type_Kinds:
An_Ordinary_Fixed_Point_Definition
A_Decimal_Fixed_Point_Definition

16.14: Change to:

Definition expects an element that has one of the following Definition_Kinds:

A_Constraint that has one of the following Constraint_Kinds:
A_Digits_Constraint
A_Delta_Constraint

A_Type_Definition that has one of the following Type_Kinds:
A_Floating_Point_Definition
An_Ordinary_Fixed_Point_Definition
A_Decimal_Fixed_Point_Definition

16.18.1: is OK as is.

16.21, 16.22, 16.33: drop "one of" from all of these.

16.30: drop "one of" from the first and third items.

16.31: Remove extra nesting.

16.34: drop "one of" from the parameter.

18.2: drop "one of"s.

18.3: Get rid of level and "one of".

18.7: Two "one of"s, get rid of level.

18.8: Drop "one of" in the parameter; correct "following" and drop the extra "of" in the return.

18.11: Drop level and remove two "one of"s.

Done with problematic parameters or results, returning to the regularly scheduled changes.

17.8: Remove the first expects line and the exception line that follows it.

19.1: OK as is (there is no "use subprogram").

20.13 -- 20.17 "; otherwise returns False."

20.28: Should raise an appropriate exception.

20.1: Delete the first sentence (but keep the index entries). Add "The type Line represents ..." The third sentence of the second paragraph is junk (of course it is supported). Replace the last sentence: "Nil_Line is the value of a default-initialized Line object."

Delete the text about undiscriminated private type.

21.1: Delete the first sentence.

"Nil_Id is the value of a default-initialized Id object."

Delete the last paragraph. The second paragraph: "The type Id represents permanent ...".

We wonder if there are other places that talk about "uninitialized" rather than "default-initialized". We do a search.

Uninitialized in 22.10: This is Data_Decomposition. Leave alone.

Uninitialized in 8.11, 8.12, 8.13, 8.14, 8.4, 8.6, 8.9. All of these are talking about contexts, not initialization.

20.7: Drop "Span is" and "that" from the first sentence, italicize "single text position" and index that. (It is a definition.) The "position within the compilation unit" in this sentence should be "location within the compilation." "Type {S}[s]pan..." in the third paragraph.

21.2: You should be able to hash a Nil_Id. So it should work for all Ids. That should be mentioned explicitly, and of course the entire definition needs to be written. The hash should be the same if Is_equal = True.

21.3, 21.4: Better explain what ordering is intended. Steve wonders if they can be incomparable (A < B = False and A > B = False and A = B= False). No, we think.

"Physical" is a bad word. Try to eliminate that word, dropping the word, or adding "version". In 21.6, it should say "Returns True if Left and Right represent the same Id, from the same version of a compilation unit."

21.8: Is OK (an exception is raised); surely Nil_Id is not represented in The_Context. "...not available th{r}ough The_Context..."

"Is_Identical (Create_Element (Create_Id(E), Enclosing_Context(Enclosing_Compilation_Unit(E))), E) is True for any element E."

"If Create_Element is called twice with the same Id and context, the two results are Is_Identical."

In the second paragraph, change to "(can be referenced through)"

21.9: "...associated with the Id including for Nil_Id".

The title should be (id) not (ids).

22: "may exist" should be "is optional". Drop the second paragraph. Get rid of the extra comma. Drop the deleted misspelling (this was new text, don't need to save corrections to it).

"Support for data decomposition is optional. The library package Asis.Data_Decomposition shall exist for an implementation that supports data decomposition. If it exists, the package shall provide interfaces equivalent to those described in the following subclauses."

Otherwise, it is OK to forget about Nil.

Greg will create wording sections for all of this.

Approve intent of SI: 9-0-0.

SI99-0038-1/01 Generic units need to be included in Is_Body_Required

Fix spelling of libaray in question.

Erhard is worried about the result for local units, but those cannot be passed here (this takes a Compilation_Unit). There isn't a way to find out whether a non-library body is required in the syntactic subsystem. But that can be done in the semantic subsystem (use Requires_Completion on a program unit view).

Erhard says that the wording says that a library subprogram specification return False. That's weird.

So add subprograms to this list.

Add "a subprogram_declaration, a generic_subprogram_declaration,". And add "Otherwise, it returns False."

This really should be consistent with Corresponding_Body.

"Returns True if the argument Compilation_Unit exists and requires a body to be present to make up a complete partition; otherwise returns False."

Approve SI with changes: 9-0-0.

SI99-0039-1/01 Change representation clauses to aspect clauses

There are two mdashs here that appear as weird characters on some machines, replace with a regular hyphen.

The question should say that the Corrigendum (rather than the Amendment) introduced the term aspect clause.

15.4 "{an}[as]"

15.31 and 15.33 are backwards (the replacement is first, the old one is second). Reverse them.

The discussion is missing a "not": If we did not make these changes, these routines would {not} appear to be usable for operational aspects, while the intent is that all aspects are included.

Approve SI with changes: 9-0-0.

SI99-0040-1/00 Various questions

Question 1: Can an instance with a formal type parameter be primitive? No, because it freezes just before the instance is declared.

Jean-Pierre's example could be non-pathological if the generic body can be imported.

Steve Baird says ASIS should allow this sort of case, since ASIS shouldn't be analyzing the language and deciding whether some case is a pathology. The only question is whether it is possible. Users would expect to be able to ask this question of any subprogram.

The group agrees that this should be added. Jean-Pierre will take an action item to write up a real SI to make this change.

Question 2: This Corresponding_Parent_Subtype is rather redundant, Parent_Subtype_Indication gives the appropriate information without losing anything. This routine seems pretty much useless (it is the combination of two other ASIS routines), and it doesn't return anything useful in many cases. So it probably should be obsolescent.

But even if we make it obsolescent, we need to say something about what it does. It should be defined to return Integer for Integer'Base. No, we should say it is implementation-defined (we don't want to make implementers any work to make it work in a particular way – it exists only for compatibility with old programs, and for that, whatever it was doing in the old implementation is best).

Jean-Pierre Rosen will take an action item to write up a real SI with this resolution.

Question 3: This is a very strange function, as it would lose some constraints (or return Nil for things that have perfectly useful values). It would be useful to give you the type, but it doesn't do that. What does it do for anonymous arrays?

The example in the question should say "For Corresponding_Type_Expression(Var), the type declaration ..."

We don't want to change the meaning of this function, because it would be hard to find bugs caused by such a change. Probably we should define that things that don't have definitions would give the first thing that does have a definition: such as Integer'Base would return Integer.

The real answer would be found in the semantic subsystem.

So we want this defined as the nearest named subtype if that exists, and nil-element otherwise. Jean-Pierre will write that up.

Approve intent of SI: 9-0-0.

SI99-0041-1/01 Adopt UTF-16 for ASIS

This is necessary to represent program text in Ada returned by ASIS; such text may include characters that need more than 16 bits and thus don't fit into a Wide_String. UTF-16 encoding is nice because it has no impact on (legal) BMP characters (there are holes for encoding purposes); they all have the same representation in a UTF-16 string representation as they do in a Wide_String.

UTF-16 is defined ISO/IEC 10646-1 Annex Q, the wording needs to say that. That standard needs to be mentioned in the references section of the ASIS standard.

Jean-Pierre Rosen will take an action item to look at adding Ada functions (for the Ada standard) to convert between UTF-8 and UTF-16 and Wide_String/Wide_Wide_String).

Change the summary to: "Wide_Strings used to represent program text that includes characters outside of the Basic Multilingual Plane should be interpreted as UTF-16."

"This has no effect for program text that only contains legal BMP characters (UTF-16 encoding is identical to the original codepoints for legal Wide_Characters)."

Approve SI with changes: 6-0-3.

Detailed Review of Ada 2005 AIs

AI05-0001-1/02 Bounded Containers and other container issues

Bounded Vectors

These containers cannot be controlled, because Ada.Finalization is Preelaborated and these packages are Pure. There probably should be an AARM implementation note to that effect.

[Editor's Note: Because of the above, wording needs to be added to say that type Vector does not need finalization in the bounded case. Otherwise, we'd inherit that wording from A.18.2(84/2), and we don't mean it.]

The assignment example is a Bounded Error, which needs to be described at the end of the section (under the Bounded Error heading). We don't do bounded errors in the normal flow of text.

Replace "The semantics of Assign are as follows:" Say something like "For procedure Assign, ..."

What exception should be raised when the capacity is exceeded: Storage_Error, Constraint_Error, or a new exception Capacity_Error? Matt would prefer Storage_Error, that would be consistent with unbounded forms. Steve Baird really hates the use of Storage_Error; it can occur anywhere. Jean-Pierre notes that Storage_Error suggests that it just runs out of memory, while other exceptions usually suggest a bug.

Tucker suggests that this capacity violation could be usefully handled, but handling Constraint_Error is dubious. He suggests that a high-level handler for Constraint_Error would make no sense (it would catch and hide all manner of bugs as well as the intended cause). He suggests that when converting from an unbounded form to a bounded form, you would not want to put a handler around every operation. But that would be necessary with either Constraint_Error or Storage_Error (they both have many unrelated causes).

Ed says that Capacity_Error sounds right. Straw vote: Capacity_Error 10, anything else 1. Matt continues to grumble, but the vote is clear.

The exception Capacity_Error will be declared in Ada.Containers. Should this be renamed in the children? (Similarly to the way IO_Exceptions are renamed in Text_IO.) The user of the instance would have to look somewhere else to find it without the rename. But those renames are bad for use clauses. Jean-Pierre claims that usually you would have withed Ada.Containers to get the generic to create the instance, so it is only an issue for library-level instances. Those are rarer for the containers than for other kinds of generics. So we won't have the renames.

Back to Assign. Instead of saying "appends", it should say "assigns the elements in the same order". We don't want to require a particular order of assignment of the elements. (Full array and slice assignments should be allowed, as well as element-by-element.)

Add to Assign: "If the check fails, Target is not modified."

For Copy, If the Capacity is non-zero but not big enough for the contents of the Source, this should raise Capacity_Error. Matt points out Reserve_Capacity does not require shrinkage, so it would be OK in this case. The group thinks this is more like Set_Capacity, you are asking for a specific size. It would be weird to get some other size.

Back to Assign again: We have a long discussion about whether containers are usable after an assignment exception. We don't want to worry about that (the object is abnormal).

"Each element of Source is assigned to the matching element of Target." 'Matching' is defined for arrays (4.5.2). (There probably should be an AARM note mentioning that.)

Assign should tamper with cursors. We should add Assign and Copy to the unbounded forms, so we should add this bullet to the tampering definition in A.18.2 (after paragraph 93):

- it calls Assign with V as the Target parameter;

Why would you want to call Copy for objects with the same capacity? If you default the capacity, it gives you an object with exactly the right capacity. That seems useful.

For Copy, it is not clear what the unbounded form should do. Should it have a capacity parameter?? Matt will decide something in his next draft. It is noted that if Assign can be written in terms of Reserve_Capacity, it would not even need to be mentioned in the bounded form.

"The function Copy returns a bounded vector object whose capacity is Source.Length if Capacity is zero, other the capacity of the return object is Capacity. If the Capacity is non-zero and less than Source.Length, Capacity_Error is raised. Each element of Source is assigned to the matching element of the return object."

(But this should be wordsmithed.)

Insert is defined in terms in Reserve_Capacity, so that would raise the exception Capacity_Error appropriately. We don't need additional wording for that.

What if the index_subtype has fewer elements than the capacity? One could try to have a subtype of Count_Type that would restrict the maximum capacity.

This seems like it could happen for the unbounded case. We want to allow indexing into the "internal array" with the Index_Subtype, and implementations might raise an exception if too long. So we should define a subtype of Count_Type, Capacity_Subtype. It would have the maximum size based on the index_subtype and the bounds of the Count_Type. It should be used as the parameter of Reserve_Capacity and as the subtype of capacity subtype and the result subtype of Capacity, and the Capacity parameter of Copy. [Should it be used as the subtype of the discriminant as well? The meeting notes don't say – Editor] This is another change that will apply to both the unbounded and bounded forms, so it shouldn't need specific mention in the bounded case.

Tucker suggests that Move for an unbounded vector say that Reserve_Capacity (Length (Source)) is called after the Clear. Then the bounded form would not need any new text (the exception would be raised appropriately).

The Implementation Advice for Move needs to be changed, however. (Some implementers want to implement all advice, we shouldn't make it impossible to follow advice!)

Set_Length doesn't need wording, it depends on Reserve_Capacity.

'Read and 'Write should only stream Length elements, not the capacity elements (which is what the default implementation would do). Matt will need to create wording for that effect for only the bounded forms. ('Read and 'Write do not include the discriminants.)

"For the procedure Reserve_Capacity, if the specified Capacity is larger than the Container.Capacity, then it raises Capacity_Error. Otherwise, the operation has no effect."

Bounded Linked List

There is no Reserve_Capacity in the unbounded linked list, so that trick for reducing the wording doesn't work for the bounded linked list. Clearly Insert will need to describe overflowing Capacity, and any other routines (like Move and Splice) that have this issue also will need to be describe that. There are three Inserts. All of these would just mention that such a check is needed; it does not need to repeat the semantics of the operations. (Repeat as little wording as possible.)

"Move" may copy the element in the bounded form; we don't want to repeat the semantics. Again, we need Implementation Advice to override that for Move from Unbounded. (That will be true for all of the containers.)

Bounded Hashed_Maps

The wording for Reserve_Capacity should be replaced for the bounded form so that it does not cause rehashing (the hash size can't change). It also needs to mention the raising of Capacity_Error, of course. (More grumbling from Matt, he still wants Storage_Error. We think he'll get over it. Or perhaps not; he seems to remember the exact meeting when we made each decision that he didn't like, and refers to the meetings repeatedly. The editor can hardly remember the locations of the meetings, much less the decisions made there, but he checked the minutes of a couple of meetings that Matt mentioned and Matt's memory is completely accurate relative to the decisions made. So perhaps "getting over it" is not going to happen real soon.)

Modulus should be defined as "the size of the primary data structure used for the hash table". Steve suggests "cardinality" of the array. Tucker prefers to just say "hash table modulus".

"Default_Modulus is an implementation-defined function of Capacity. This is used for..." By saying "implementation-defined", we are requiring documentation as to how it works.

Bounded Ordered Map

There is no Reserve_Capacity for Unbounded Ordered_Maps, so again we need a list of operations that start with a check.

Bounded Sets

The sets are the same as the maps in the interesting ways. Matt should work out the details.

Protected Queues

Is_Empty seems to have disappeared from the interface, although it is described. Put it back.

The descriptions should not say "Back of queue"; it should say "tail of queue" instead; "front of queue" should be "head of queue".

We don't want more functionality in the interface; specific types can add additional operations, and we want the interface to be minimal.

The parameter name probably should be "Container".

The name of the queues is fine as is; these will not usually be instantiated at library level.

Should these be children, or siblings with a generic formal package? The names would be more regular if they're not children:

```
with Ada.Containers.Queues;
generic
    with package Queues is new Ada.Containers.Queues (<>);
package Ada.Containers.Bounded_Queues is
   pragma Pure;

   type Queue (Capacity : Count_Type) is
     synchronized new Queues.Queue with private;

private

   ...
```

We prefer a generic formal package parameter, as given above.

Bounded_Queues and Unbounded_Queues are the names of the concrete queue packages.

Sorting

The new sort differs from the existing ones in that it does not require the name of an array type. That means it sorts "indexable containers", essentially anything that can be indexed.

Update the text defining "Less" to reflect the changes of AI05-0044-1: "strict weak ordering relationship (see A.18)".

Case-insensitive functions

Matt wonders about the Wide versions of the case insensitive functions. That is very problematical, because wide case mapping is not one-to-one character conversions. (German ß is one character, the upper case is two characters "SS", for example. Unicode documents say that it is possible to have 1 to 3 and 3 to 1 conversions!) Note that the wide character mappings are specifically defined be identity functions for non-Latin-1 characters. (See also the discussion of AI05-0127-1.)

Other Protected forms

Why no more protected forms? It's very hard to avoid race conditions and deadlock scenarios - especially for iteration (and what good is a container that does not have an iterator?). It's also the case that these containers are not potentially blocking (based on the language definition); so that containers can be used from protected operations. That means it's not that hard to make your own protected versions (and project-specific versions can put the entire iterator in the protected object, eliminating the race and deadlock problems).

AI05-0050-1/05 Return permissions are not enough for build-in-place

Should the recursive note be normative? It should be normative. Someone suggests simply adding that to the lead-in:

"For a function call used to initialize a composite object {or a return object} with a constrained nominal subtype:"

No, that doesn't work because the nominal subtype of the return object may be unconstrained. Another suggestion is:

"For a function call used to initialize a constrained composite object:"

This is confusing. Is the target constrained if the return object is constrained? We're not sure; maybe this wording doesn't work.

We take a break to look at AI95-0067-1 to see if the wording changes there help any with this AI's wording. After doing that, we return to this wording.

"For a function call used to initialize a composite object with a constrained nominal subtype or used to initialize a return object that is built in place into such an object:"

Drop the first paragraph of the AARM note.

The model is that the exception is raised at the same place it would have been if it was not built in place: the point of the assignment statement where the constraint would have otherwise failed. We don't want it moving around.

Can there be ACATS tests for this? ACATS tests could be created for those cases where the permission does not apply (specifically functions returning unconstrained definite types used to initialize constrained objects). But the permission itself isn't testable (trying to guess where an exception might be incorrectly raised seems pointless).

Approve AI with changes: 7-0-0.

AI05-0054-1/01 Variable views of constant objects
AI05-0054-2/04 Variable views of constant objects
AI05-0054-3/01 Variable views of constant objects
AI05-0054-4/01 Variable views of constant objects

Erhard explains the various alternative approaches that have been considered.

He distributes a new version of alternative 2. (Later, Tucker insists that this should treated as yet another alternative; for the rest of the minutes we'll call it alternative 4.)

Tucker says that when this happens, the library client hasn't done anything wrong, nor has the library author. It's only the combination that causes the issue. So it is annoying to make the program erroneous; the only "gain" is to allow tools to assume no changes.

Erhard suggests that we should do nothing; the library writer should not use **in** parameters for such a type, and then the user can't write "constant". But such a thing cannot be done unless you abandon functions (nothing but **in** parameters there).

Jean-Pierre suggests that we need a way to declare (visibly) that a type does not allow declaring something constant. That seems pretty heavyweight.

Looking at Erhard's alternative 4: It should say "partial view". Steve notes that an array of private types is not private, it is an array. But the rule needs to apply to it (else we have madness). This approach doesn't appear promising.

We go back to the original alternative 2 wording. Erhard's main objection is to the wording in 3.3(13) that essentially says that constants aren't constant. Tucker suggests that we rewrite this section to say that is true only for controlled or immutably limited types, thus limiting the "damage".

Ed suggests that some types are "fundamentally variable", and that variable views always can exist for such types. We're not sure that we need a term for this, although it might help the understanding.

Tucker suggests a rewording of 3.3(13):

> An object is either a constant object or a variable object. [The value of a constant object cannot be changed between its initialization and its finalization, whereas the value of a variable object can be changed.]

> Similarly, a view of an object is either a constant or a variable. All views of a constant {elementary} object are constant. {All views of a constant composite object are constant, except for parts that are of controlled or immutably limited types; variable views of those parts and their subcomponents may exist. In this sense, objects of controlled and immutably limited types are inherently mutable.} A constant view of [a variable] {an} object cannot be used to modify [the value of the variable] {its value}. The terms constant and variable by themselves refer to constant and variable views of objects.

There is nothing in the language that says that subcomponents of constant objects are "constant objects" (but they are constant views). We don't need to define constant and variable views here; those are defined later in detail, and anything we said in this paragraph would either be redundant or conflicting, so we deleted Tucker's previous changes in this area.

Steve notes that similar issues occur inside an extended return if an implementation chooses to build-in-place when it does not have to.

Erhard and Jean-Pierre will take an action item to come up with a user note for 3.3 to make it clearer what is going on.

Approve AI alternative 2 with changes: 7-0-1.

No action on alternative 1, 3, and 4: 7-0-1.

AI05-0067-1/06 Objects that are built in place

In the first paragraph of inserted text after 7.6(17), Tucker is worried about losing the connection between assignment and build in place. He suggests adding a sentence to this paragraph:

> When a function call or aggregate is used to initialize an object, the result of the function call or aggregate is an anonymous object, which is assigned into the newly-created object. {For such an assignment, the anonymous object might be "built in place"}. Under certain circumstances, the anonymous object is {required to be} "built in place", in which case the assignment does not involve any copying. In particular:

AARM discussion: "... in Ada 95.{ }We do this because..."

The AARM To Be Honest should be AARM Discussion.

7.6(19), AARM Note: "this implementation" should be "intended implementation". These are probably Implementation Notes.

Approve AI with changes: 6-0-1.

AI05-0095-1/02 Address of intrinsic subprograms

In the discussion, before the last example, "prefix [it]{is}".

The AARM note is just making sure that uses of a renaming of an entity are illegal if direct uses of the entity would be illegal.

Approve AI with changes: 8-0-1.

AI05-0099-1/02 The tag, not the type, of an object determines if it is controlled

Typo: discussion, 4th line, "descendant" is misspelled.

To be honest paragraph has a double "of".

7.6.1(8) new text: "...is [of]{a} tagged type..."

Approve AI with changes: 8-0-1.

AI05-0100-1/02 Placement of pragmas

The last sentence should be another bullet, without the Also.

- At any place where a compilation_unit is allowed.

Someone doesn't like the summary, and proposes a replacement: "Pragmas are allowed in places where removing them would not render the program syntactically illegal."

But that doesn't capture the problem (and seems to allow pragmas in other places than they are actually allowed). So reword the summary as follows:

"For lists that allow pragmas, if the list may have no elements, then the list may consist solely of pragmas."

Approve AI with changes: 8-0-1.

AI05-0101-1/03 Remote functions must support external streaming

In the question, RACW should be expanded to remote access-to-class-wide.

The new bullet E.2.2(15) should be actually after E.2.2(16/1).

The !ACATS section should mention that new B-tests are needed.

Approve AI with changes: 5-0-2.

AI05-0106-1/02 Representation items are not allowed on generic formal parameters

There is much discussion about whether this is too restrictive. We surely don't want it for any existing language-defined representation pragmas. We decided last time that we didn't want to have to define matching rules for such pragmas, and that would be necessary if we allowed them.

Add "Unless otherwise specified, ..." to the wording, so we would leave the door open for future pragmas. To do that, matching rules are required. Hopefully, any such implementation-defined pragmas will define such rules.

Better update the discussion the same way.

Approve AI with changes: 6-0-3.

AI05-0108-1/01 The incomplete view from a limited view does not have a discriminant part

Summary: "full view" should be "some view". (The limited view could have come from a partial view with discriminants.)

Wording: This is a bullet, and should be marked as such.

Ramification: "This {is} necessary..."

Approve AI with changes: 8-0-0.

AI05-0109-1/01 Impossible check in S'Class'Input

Typo in summary: "It {is} not possible..."

Add an AARM Note: "Descendant_Tag will ensure that the returned tag is covered by T'Class."

Approve AI with changes: 8-0-1.

AI05-0116-1/01 The value of Alignment for a class-wide object

Answer the first question (No.)

Replace the second part of the wording by "is the Alignment of the specific type identified by the tag of the object being created". This change is needed because the allocator could be for a classwide type with an initial value, so there may not be a specific type given in the allocator.

Add to the first sentence of the summary "For an implementation, "

Tucker and Steve would like to require that T'Class'Alignment equal T'Alignment.

Erhard suggests adding this requirement 13.3(26.4/2); he sees it as a definition. No, there should be Implementation Advice to this effect; we don't *require* anything about Alignment.

Randy thinks that since this is nonsense, we ought to make 13.3(32.1/2) a static rule, but no one wants to reopen that. We should add an AARM note suggesting that implementers that don't follow this specific advice should have their heads examined. (Because you could call inherited subprograms with less aligned objects than expected.)

Approve intent of AI: 7-0-2.

AI05-0118-1/01 How do parameter associations associate?

We do a bit of wordsmithing.

Change insertion after 6.4(9) to: "The formal parameter for a positional parameter_association is the parameter with the corresponding position in the formal part."

Change insertion after 12.3(9) to: "The generic_formal_parameter_declaration for a positional generic_association is the parameter with the corresponding position in the generic_formal_part of the generic unit being instantiated."

But it is not clear what the wording for 6.4.1(2) should be. Randy argues that there is no clear definition of how formal parameters and actual parameters are associated. That needs to be fixed and is the point of his changes to 6.4.1(2). The group eventually comes around to his viewpoint. We then spend a long time discussing where the best place for these changes is.

Change 6.4.1(2) to: "The *formal_parameter_*selector_name of a {named} parameter_association shall resolve to denote a parameter_specification of the view being called{; this is the formal parameter of the association. The formal parameter for a positional parameter_association is the parameter with the corresponding position in the formal part of the view being called}." Index named association and positional association here.

Delete the insertion after 6.4(9)

Discussion: third paragraph has no verb. "It would {look} odd..."

Approve AI with changes: 8-0-1.

AI05-0120-1/01 Is the current instance of a protected object a constant?

Add to summary "within {the body of a protected function}."

The location is just 3.3(21).

Approve AI with changes: 8-0-1.

AI05-0121-1/01 Opening a non-existent file in Append_File mode

Randy notes that the person who asked the original question on comp.lang.ada was happy with the existing facilities of Ada. The !discussion of the AI explains why (the race condition that exists in the current facilities cannot be eliminated, as it is possible for an outside actor to affect the file between the Open and Create).

No one seems interested in this idea.

No action: 7-0-1.

AI05-0123-1/01 Composibility of equality

We want to just do this for record types (really only untagged record types, since tagged records already compose). Other types have problems with compatibility with other operators, especially in generics. That's not how the AI was written up. Steve Baird will rewrite the AI with this in mind.

We'll just do composition unconditionally; the AdaCore survey showed no cases in their system that would change incompatibly (but one case where a bug would be fixed). There aren't any ACATS tests, either, according to their survey.

Steve needs to worry about the useless case of abstract equality operations. He thinks of Program_Error, but Tucker worries that that would be unacceptably incompatible. Probably, we'll just have to revert to the original operation in that case. (But that means that specifying an abstract equality as ASIS does is actively harmful!)

AI05-0126-1/01 Dispatching when there is no inherited operation

7.3.1(6/3) says that all primitive subprograms are inherited, but they might not be declared.

But this wording talks about implicit declaration of operations, not inheritance of operations. (Randy's AARM note needs to be rewritten to talk about declaration and existence, not inheritance.) The title of the AI needs to be changed.

Tucker would prefer to restructure this and add a sentence after the bullets:

"For the execution of a call on a dispatching operation, the action performed is determined by the properties of the corresponding dispatching operation of the specific type identified by the controlling tag value:

- if the corresponding operation is explicitly declared for this type, [even if the declaration occurs in a private part], then the action comprises an invocation of the explicit body for the operation;
- if the corresponding operation is implicitly declared for this type and is implemented by an entry or protected subprogram (see 9.1 and 9.4), then the action comprises a call on this entry or protected subprogram, with the target object being given by the first actual parameter of the call, and the actual parameters of the entry or protected subprogram being given by the remaining actual parameters of the call, if any;
- otherwise, the action is the same as the action for the corresponding operation of the parent type."

The AARM Note becomes a ramification.

Steve Baird wonders about the following:

```
package P is
   type Intf is limited interface;
private
```

```
   procedure Hidden is null;
end P;

with P, Ada.Finalization;
package Q is
   type Ugh is new Ada.Finalization.Controlled and P.Intf;
end Q;
```

Hidden is not declared in this case, and the parent doesn't have Hidden. After discussion, it becomes obvious that this is a problem with all null procedures of interfaces, they don't need have to be hidden to cause a problem.

So we need some wording about looking at the progenitor from which it is inherited.

So change the last bullet to:

- Otherwise, the action is the same as the action for the corresponding operation of the parent or progenitor type from which the operation was inherited.

The call in the example in the question of the AI should be `P.Oper (X)`

Approve AI with changes: 8-0-0.

AI05-0127-1/01 Locale consideration for To_Upper and To_Lower

We surely do not want to change the way any existing functions work, that would be unacceptably incompatible.

There might be value to having some locale-specific package with Wide to Wide conversions. Something like Ada.Characters.Localized_Handling.

There is a lot more than just conversions that would make sense as locale-specific operations.

Brad will try to make a proposal for a new package to support such facilities.

Keep proposal alive: 8-0-0.

AI05-0128-1/00 "/=" is a primitive operation

Erhard and Randy are worried if saying primitive would cause some problem. Tucker says "/=" is inherited. Randy is dubious, 6.6(6) says "a declaration", not "an explicit declaration". That means it includes implicit ones, so the implicit "/=" is implicitly declared for each type.

The solution seems to be add text to 6.6(6) to say that the "/=" is primitive if the "=" is primitive. And add a To Be Honest to 3.2.3 to make it clear that "/=" is primitive even though it is not explicitly declared or inherited.

Fix typo in summary.

Tucker will take this one to do on his next train ride.

Approve intent of AI: 9-0-0.